Workflow Patterns as a Means to Model Task Succession in Games – A Preliminary Case Study

Simone Kriglstein[1], Ross Brown[2], and Günter Wallner[3]

[1] Vienna University of Technology, Vienna, Austria,
kriglstein@cvast.tuwien.ac.at
[2] Queensland University of Technology, Brisbane, Australia,
r.brown@qut.edu.au
[3] University of Applied Arts Vienna, Vienna, Austria,
guenter.wallner@uni-ak.ac.at

Abstract. Over about the last decade, people involved in game development have noted the need for more formal models and tools to support the design phase of games. It is argued that the present lack of such formal tools is currently hindering knowledge transfer among designers. Formal visual languages, on the other hand, can help to more effectively express, abstract and communicate game design concepts. Moreover, formal tools can assist in the prototyping phase, allowing designers to reason about and simulate game mechanics on an abstract level.
In this paper we present an initial investigation into whether workflow patterns – which have already proven to be effective for modeling business processes – are a suitable way to model task succession in games.
Our preliminary results suggest that workflow patterns show promise in this regard but some limitations, especially in regard to time constraints, currently restrict their potential.

Keywords: Game Design, Design Tools, Workflow Patterns

1 Introduction

Over about the last decade, people involved in game development have repeatedly voiced the need for more formal models and tools for designing games. The traditional game design document, a detailed – mostly textual – description of all aspects of a game design, and still the main design reference (cf. [1]) has been criticized for several drawbacks, among others, to be time-consuming to create while at the same time being rarely read by members of the development team (see, e.g., [2]) and to need constant maintenance to keep the information up-to-date with the highly iterative game development process (cf. [2, 3]). Kreimeier [4], in a survey of game design methods, noted that the informal discussion of game design makes it difficult to put *individual insight into a context of established knowledge*. Therefore, formal visual languages have been proposed to more effectively express, abstract and communicate gameplay concepts or relationships between them (e.g., [2, 5–7]).

In a related context, Nelson and Mateas [8] point out that game development is lacking tools to visualize and reason about systems of game mechanics on an abstract level. However, game designers agree upon that prototyping is essential for verifying design ideas (cf. [1, 9, 10]). Games are complex and dynamic systems and prototypes help to verify design assumptions and understand emergent properties before actually going into production. Yet, prototyping can be a time-consuming and costly process. On that account, researches have therefore proposed prototyping tools to visually model and simulate different aspects of game design on an abstract level (see, e.g., [3, 11, 12]).

In this paper we present a preliminary investigation into whether the widely supported original workflow patterns [13, 14] are a suitable way to model the succession of tasks players have to perform in a game. In a similar manner to Van Der Aalst et al. [13], we have sought to use a pattern analysis approach to identifying recurring structures within lists of activities performed by players in computer games. Such a pattern approach provides independence from the underlying game implementation environments, and the underlying theoretical languages used to describe such flows of activities. A workflow patterns approach enables us to specify abstractly the tasks to be performed by players within game environments, as the tasks given to players may share similarities in structure to tasks in business systems. The future benefits of such an approach are thus the ability to define such gameplay activity patterns in a rigorous and formal manner (drawing from the Petri net formalisms of workflow patterns) giving us both the power to deeply analyze their correct execution, and then to execute them in many systems capable of supporting Petri net based formalisms.

2 Related Work

To date a lot of work has been conducted to establish a common design vocabulary, for instance, in form of design patterns in order to be able to *express clearly intentions, analyses, and opinions regarding gameplay* [15] – too many to cover here in detail. An extensive collection of design patterns as well as references on the topic can be found at the *Game Design Patterns 2.0* website [15].

While design patterns are usually expressed in natural language, some people involved in game development have argued for more visual models to more clearly and compactly express and communicate ideas about game design (see the recent survey of Almeida and Silva [1] for a discussion on this subject). For example, Koster [5] presented a graphical notation system as a complement to classic game design documents to better communicate the game design. Bura [6], building upon the work of Koster [5], proposed a visual grammar which was influenced by Petri nets [16] and resembles data-flow representations of processes (whereas we are concerned with a control-flow perspective). Araújo and Roque [11] also applied Petri nets to model games and emphasized that the formal semantics of Petri nets can be leveraged to formally analyze and simulate game design elements already in an early design stage (e.g., to detect unwanted behavior).

Recently, Dormans [3] developed the *Machinations* tool, a graphical notation framework to express the rules of and simulate game economies in order to, for example, balance the game's economy or to prevent dominant strategies. While borrowing concepts from Petri nets, *Machinations* diagrams are aimed to be less complex and more accessible to designers. The work has been later extended by Klint and van Rozen [12] to allow not only for simulation but also for formal analysis of game designs. Adams and Dormans [17] compiled a list of useful design patterns which can be used with the *Machinations* approach. While the above mentioned works aim at expressing game mechanics, Cook [7] focused on the player experience and proposed *skill chain diagrams* to visually represent how players learn and acquire skills in a game.

Workflow concepts themselves have already been applied to games by Brown et al. [18] before – although from a more technical point-of-view as being used in this paper – by using a workflow language to control a distributed game and to script game tasks.

3 Use Cases

For our initial investigation we have chosen two games that can be loosely classified as action games but are still distinct in terms of gameplay. In particular we have chosen *Need for Speed: The Run* [19] (NFS) and *Half Life 2* [20] (HL2). NFS is an action racer where the player takes part in a race across the US. The complete race is broken down into stages with each stage again divided into different race events. HL2 is a single player first-person shooter (FPS), incorporating a science fiction narrative revolving around a dystopian future scenario of battle with an evil multidimensional empire known as the *Combine*. Like a number of first person shooters, you play a protagonist, in this case *Gordon Freeman*, and acquire weapons and health packs in order to defeat non-player characters (NPCs) and traverse levels.

3.1 Need For Speed: The Run (NFS)

The race events in NFS can roughly be divided into: a) checkpoint races, b) sprint races, c) rival races, and d) battle races. In case of a checkpoint race, a sequence of checkpoints has to be crossed. In workflow terms this can be described with the *Sequence* pattern (Workflow Pattern #1 [22]). This is illustrated in Figure 1a.

A bit more complex is the workflow pattern for the sprint race event (see Figure 1b), where the player has to overtake a number of opponents and to cross the goal line in first position. The task to overtake multiple opponents can be described as a cycle that repeats until the player is on the first place (WP #10 - *Arbitrary Cycles* [22]). However, overtaking all opponents and reaching the first place is not enough to win the race, the player also has to maintain the lead until the goal line is reached. This is modeled with a second arbitrary cycle. The

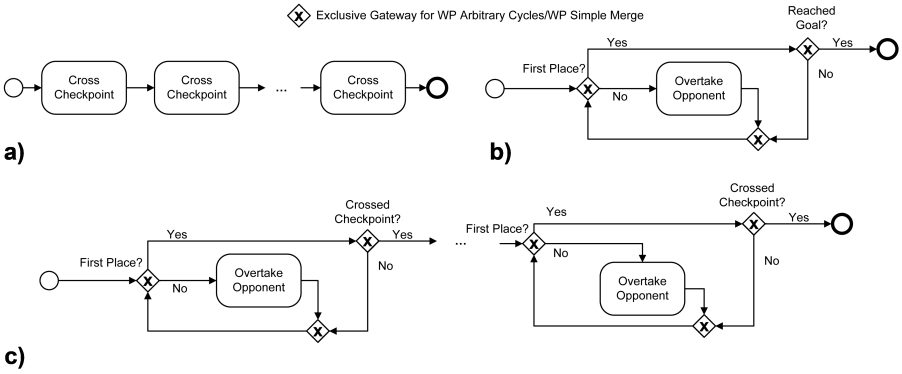

Fig. 1. The general three workflow patterns for the four main race events of NFS illustrated using the BPMN 2.0 visual grammar [21]: a) Pattern for checkpoint race event, b) pattern for sprint and for rival race event, and c) pattern for battle race event.

Fig. 2. Examples of the major tasks performed in an FPS – from left to right: `Kill Agent`, `Use Resources` (ammunition, healing items) and `Open Door`.

Simple Merge pattern (WP #5 [22]) is used to merge the outgoing branch of the `Overtake Opponent` node and the false branch of the second arbitrary cycle.

The rival race event is technically the same as the sprint race event except that the player has to race against specific drivers that are part of the storyline. It can therefore be modeled the same way as the sprint race.

The battle race is a slight modification of the sprint race, where the player has to overtake an opponent and keep the lead until a checkpoint is reached after which the player races against another opponent until the next checkpoint. This repeats several times during a single battle race. The basic structure of the workflow pattern for a battle race event is therefore similar to the workflow pattern for a speed/rival race event with the difference that it repeats several times in sequential order.

3.2 Half Life 2 (HL2)

General analysis of the levels in the game indicate a number of workflow patterns from a level designer perspective. First of all, entrance to a level triggers

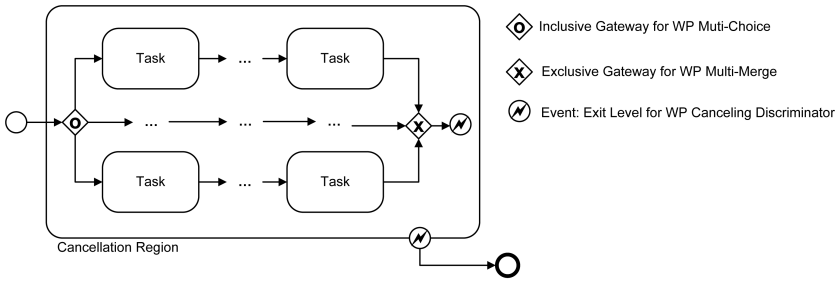

Fig. 3. General FPS workflow pattern for HL2.

the instancing of multiple tasks (WP #6 - *Multi-Choice* [22] in combination with WP #8 - *Multi-Merge* [22]) to be performed in the game. We group these into three major task types, viz. `Kill Agents`, `Use Resources`, `Open Doors` (see Figure 2). `Kill Agents` is the destruction, by whatever means, other agents in the game. `Open Door` tasks are a general description of having to remove a physical impediment to progress spatially, and may take the form of, for example, opening a door, destroying an oil drum, or killing a blocking NPC. These sequences are, however, embedded within a set of multiple tasks that can be potentially ignored. For example, you do not have to kill all agents to progress through a level. This killing of agents and collection of resources is often up to the discretion of the player. There are elements in the game that form bottlenecks, typically entrances with doors, that must be traversed before the gameplay continues.

A subset of the tasks require a certain sequence (WP #1 [22]) for them to be completed. In the level *Routekanal1* there are certain sequences that must be performed before progression can occur. For example, you clear out an area of NPCs with a mounted machine gun (`Kill Agents`) and come to an oil drum that has to be destroyed (`Open Door`) before you can traverse to the next location and enter a tunnel. Note, it is possible to avoid shooting the NPCs if you are skilled enough in traversal, but you must shoot the oil drum, thus this is an example of an enforced sequence within multi-choices.

Once these tasks are completed, and the level traversed, a final door opening task is completed to exit the level. When this happens, the rest of the tasks are removed from the list allocated to the player. You cannot go back to the level to finish the other remaining tasks, you must recommence the level in order to begin again, or, you must go to a save point in the game to recommence the level at the same state. This is an example of the *Canceling Discriminator* pattern (WP #29 [22]).

Combining these four workflow patterns together, we construct an overall FPS workflow pattern shown in Figure 3. We argue that this pattern, in slight variations, forms the structure of a number of FPS games. Thus designs for other FPS games can be configured from an executable form of such a pattern.

In addition, there is further detail with the tasks being performed in the cancellation region. In a similar manner, such tasks may be devolved into a split

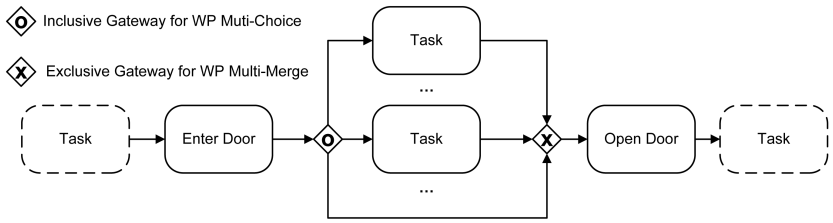

Fig. 4. Exploded BPMN view of tasks in Figure 3, illustrating how the door opening tasks are key to enforcing particular orderings of tasks in HL2 as an FPS.

upon opening a door and entering a region of a level. We define that a level is broken up into regions of gameplay, with door opening choke points, that enforce a sequence, but contain within them multi-choices and merging, as per Figure 4. Note that this pattern has no cancellation upon entering a new region. So the tasks in the split can be circumvented at the discretion of the player.

4 Discussion

In the presented use cases we focused on the description of the succession of tasks which players have to perform to successfully complete a level but we did not consider implications of failure. However, modeling the consequences of failing a task can be equally important in game design (e.g., what happens if a checkpoint in a racing game is not reached). Workflow patterns may also be suitable for this task, for example, the WP #4 - *Exclusive Choice* [22] can be used to describe branching depending on if the previous tasks were completed successfully or not.

Workflow patterns can be used to describe tasks on different levels of abstraction (e.g., opening a door could be described simple as `Open Door` or in more detail by modeling what actions are necessary to actually open it, for instance, `Pull Lever` followed by `Turn Knob`). Furthermore, a combination of the different abstraction levels can be used, for example, to provide a high-level overview with more detailed descriptions on demand. For example, Figure 3 shows an overview of the overall structure of tasks in HL2 while Figure 4 provides a more detailed description of how the tasks itself are actually modeled.

One of the biggest limitations of the original control flow workflow patterns [13, 14] for modeling task succession is perhaps their limited support of time aspects, e.g., time spans or timeouts. However, time aspects play a very important role in games. For example, in the above NFS use case the player has to reach the checkpoints in a predefined time limit. Fortunately, several extensions to the original workflow patterns have been proposed recently to handle task related time constraints (e.g., [23, 24]). Future work could therefore focus on analyzing how such time patterns can be used for the description of time critical tasks in games.

Lastly, modeling tasks in a specific modeling language needs specific knowledge in order to understand the notation. In our previous work (cf. [25]) we

therefore presented a storyboard approach to annotate workflows with images to provide additional support for different stakeholders to understand and discuss workflows independently from their experience with the actual modeling language. Since storyboarding is commonly used to support level design in games (see, e.g., [26]) a possible direction for future work could be to adapt this approach for modeling task succession.

5 Conclusions

In this paper we investigated the applicability of workflow patterns for modeling task succession in games. For this purpose we have chosen two different games as initial case studies. Our first results suggest that workflow patterns show certain promise but that there are also some limitations that currently constrain their potential in regard to games and which need further investigation, in particular, the ability to model winning and loosing conditions and time aspects. Further case studies with different games will be necessary to confirm our preliminary observations.

References

1. Almeida, M.S.O., Silva, F.S.C.: A systematic review of game design methods and tools. In Anacleto, J., Clua, E., Silva, F., Fels, S., Yang, H., eds.: Entertainment Computing ICEC 2013. Volume 8215 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2013) 17–29
2. Librande, S.: One page designs. Presentation at the Game Developers Conference. (2010)
3. Dormans, J.: Engineering Emergence: Applied Theory for Game Design. PhD thesis, Amsterdam University of Applied Sciences (2012)
4. Kreimeier, B.: Game design methods: A 2003 survey. Available at: `http://www.gamasutra.com/view/feature/2892/game_design_methods_a_2003_survey.php` (2003) Accessed: February, 2014.
5. Koster, R.: A grammar of gameplay: game atoms: can games be diagrammed? Presentation at the Game Developers Conference (2005)
6. Bura, S.: A game grammar. Available at: `http://www.stephanebura.com/diagrams/` (2006) Accessed: February, 2014.
7. Cook, D.: The chemistry of game design. Available at: `http://www.gamasutra.com/view/feature/129948/the_chemistry_of_game_design.php` (2007) Accessed: February, 2014.
8. Nelson, M.J., Mateas, M.: A requirements analysis for videogame design support tools. In: Proceedings of the 4th International Conference on Foundations of Digital Games, New York, NY, USA, ACM (2009) 137–144
9. Neil, K.: Game design tools: Time to evaluate. In: Proceedings of 2012 DiGRA Nordic, University of Tampere (2012)
10. Dormans, J.: The effectiveness and efficiency of model driven game design. In Herrlich, M., Malaka, R., Masuch, M., eds.: Entertainment Computing - ICEC 2012. Volume 7522 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2012) 542–548

11. Araújo, M., Roque, L.: Modeling games with petri nets. In Atkins, B., Kennedy, H., Krzywinska, T., eds.: Breaking New Ground: Innovation in Games, Play, Practice and Theory: Proceedings of the 2009 Digital Games Research Association Conference, Brunel University (2009)
12. Klint, P., van Rozen, R.: Micro-machinations: A DSL for game economies. In Erwig, M., Paige, R.F., van Wyk, E., eds.: Proceedings of the International Conference on Software Language Engineering (SLE 2013). Volume 8225 of Lecture Notes in Computer Science., Springer (2013) 36 – 55
13. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distrib. Parallel Databases **14**(1) (July 2003) 5–51
14. Russell, N., ter Hofstede, A., van der Aalst, W., Mulyar, N.: Workflow control-flow patterns: A revised view. Technical Report BPM-06-22 (2006)
15. Björk, S.: Game design patterns 2.0. Available at: `http://gdp2.tii.se` Accessed: February, 2014.
16. Petri, C.A.: Kommunikation mit Automaten. PhD thesis, University of Bonn (1962)
17. Adams, E., Dormans, J.: Game Mechanics: Advanced Game Design. 1 edn. New Riders Publishing (2012)
18. Brown, R., Lim, A., Wong, Y., Heng, S.m., Wallace, D.: Gameplay workflow: A distributed game control approach. In: Proceedings of the 2006 International Conference on Game Research and Development. CyberGames '06, Murdoch University (2006) 207–214
19. EA Black Box: Need for Speed: The Run. [Xbox 360] Electronic Arts (2011)
20. Valve Corporation: Half-Life 2. [PC] Valve Corporation (2004)
21. Object Management Group: Business process model and notation version 2.0. Available at: `http://www.omg.org/spec/BPMN/2.0/` (2011) Accessed: April, 2014.
22. Workflow Patterns Initiative: Control-flow patterns. Available at: `http://www.workflowpatterns.com/patterns/control/` (2010) Accessed: April, 2014.
23. Lanz, A., Weber, B., Reichert, M.: Workflow time patterns for process-aware information systems. In Bider, I., Halpin, T., Krogstie, J., Nurcan, S., Proper, E., Schmidt, R., Ukor, R., eds.: Enterprise, Business-Process and Information Systems Modeling. Volume 50 of Lecture Notes in Business Information Processing. Springer Berlin Heidelberg (2010) 94–107
24. Niculae, C.: Time patterns in workflow management systems. Technical Report BPMcenter.org (2011)
25. Nardella, K., Brown, R., Kriglstein, S.: Storyboard augmentation of process model grammars for stakeholder communication. In: Proceedings of International Conference on Information Visualization Theory and Applications, SciTePress (2014) 114–121
26. Pizzi, D., Lugrin, J.L., Whittaker, A., Cavazza, M.: Automatic generation of game level solutions as storyboards. Computational Intelligence and AI in Games, IEEE Transactions on **2**(3) (2010) 149–161